B.A.C.K. User Guide

version 0.3 - August 10th, 2008

1 Index

1. What is B.A.C.K.?
2. Required software
3. How to configure B.A.C.K.
4. How to create a new passport for the simulator
5. How it works
6. Bugs
7. Fix
8. Next updates
9. Project members
10. Special thanks

2 What is B.A.C.K.?

B.A.C.K. is a python software that accesses the new generation e-Passports. B.A.C.K. uses the known information about passport's owner to attempt finding the character sequence used by the Basic Access Code protocol – the Machine Readable Zone (MRZ). In case of success, B.A.C.K. reads the entire content of the passports.

The operations can be performed both locally and remotely, by means of a system that simulates the passport on a XML file.

3 Required software

The followed components are required to run B.A.C.K.:

* Python – for the language interpreter used by the libraries
* pylab – for the primitives of numerical calculus
* pycrypt – for the cryptography basic modules
* Imaging – for images view
* pySerial – for ACG or Frosch readers
* pyscard and PCSC-Lite – for the PC/SC support
* OpenSSL – to extract the public keys from the e-Passports

4 How to configure B.A.C.K.?

B.A.C.K. can be configured to work both on a real reader connected to a passport and in simulation mode.

Real reader mode To configure B.A.C.K. to work as a real reader, open the RFIDIOtconfig.py file from the Client directory, and check that the line relative to your reader from the readers section is uncommented.

Simulation mode To configure B.A.C.K. to work in simulation mode, open the RFIDIOtconfig.py file from the Client directory, and uncomment only the line readertype=RFIDIOt.rfidiot.READER_SOCKET from the readers section. Moreover, B.A.C.K. allows two access modes to the reader: local mode and remote mode.

Local access mode To use the application in local access mode, open the RFIDIOt.py from the Client folder, find the "self.socket.connect" string and check that the IP address is 'localhost'.

Remote access mode To use the application in remote access mode, open the RFIDIOt.py from the Client folder, find the "self.socket.connect" string and insert the server address (e.g., 'myhost.mydomain.com').

5 How to create a new passport for the simulator

To create a new passport for the simulator, you need to create a XML file containing the passport data (they can be found by invoking the mrpkey.py module).

Moreover, you have to copy and rename one of the XML files from the Server/repository folder. Then open the new file with any text editor and modify the content as it follows:

* The EF_SOD part is common for all the passports so there's nothing else to do
* The EF_SOD has as value attribute the first 8 digits of the string that contains the data of the digital certificate. The other ¡data1¿...¡data15¿ are the remaining part of the certificate, divided in 236 characters strings
* On the EF_DG1 part insert the data about the passport holder (the value attribute is common for all passports)
* The EF_DG2 part contains the data about the image of the document holder, stored on the several ¡data¿ as groups of 236 digits. The value attribute is composed by the first 8 digits of the string that represents the image.

Remember the XML file needs to be stored on the 'Server/repository' folder.

6 How it works

B.A.C.K. is made by two components: the passport (named Server) and the user program (named Client).

Server Command: python directory_B.A.C.K./Server/Server.py xml_file_name.

The Server invokes the XML parser on the xml_file_name that simulates the passport and extrapolates the data structure whose fields reflect the passport content. Then the Server waits until a Client establishes a connection. It is not necessary to invoke the Server if B.A.C.K. is used in real reader mode.

Watch out! The Server start-up module must be implemented on the destination machine. This means that if the target is 'cipro.dia.unisa.it' the Server procedure must be invoked on the related machine. If the target is 'localhost', then you need to invoke the procedure on your own machine.

Client Command: python directory_B.A.C.K./Client/main.py [MinPassNum MaxPassNum MinBirthDate MaxBirthDate Sex=[M—F] MinExpiryDate MaxExpiryDate]

The Client makes a connection to the Server. In case the parameters MinPassNum MaxPassNum MinBirthDate MaxBirthDate Sex=[M—F] MinExpiryDate MaxExpiryDate are passed, the Client will start a procedure of exhaustive search to obtain the MRZ of the passport on the base of the inserted data. If the

parameters are not passed from the command line, the application will ask the user to insert them one by one. On both cases data always go through syntactic and semantic controls.

7 Bugs

Some problems were encountered during software development:

* The window related to owner's data does not display the image: the problem is caused by the specific format of the image stored inside the passport (JPEG2000)
* The estimation of the maximum execution time refers to the machine that was used for the software testing (simulated reader mode) – HP Compaq nx6125, AMD Turion 64 Mobile ML-32 CPU, RAM 879 MB
* The application does not point out any HW/SW change (included the passage from real reader to simulated reader) that could affect the effective execution time. This entails the absence of the update of the maximum execution time.
* In case of a severe error, the server could crash and disconnect. A Server restart is needed on such cases.
* The application was not tested for multiple accesses. Please point out any bug.

8 Fix

* To update the estimation of the maximum execution time you need to launch the Client application and attempt a number of checks. At the end of the execution, the number of checks per minute is calculated by applying the following formula:

num_checks_per_minute = (60 x num_checks_exec) / exec_time

Once the number of checks per minute has been calculated, open the Client/main.py file with a text editor, find the string 'XXX' and update the actually calculated value.